2.11 DECISION-MAKING LOGIC PROCESSES

The representative logic processes or functions within a mission-level combat model are maintaining and developing the tactical situation, assessing the situation, making decisions based upon the assessment, and selecting actions. In SWEG, these four functional areas are implemented by the five cognitive processes notice, digest, react, ponder, and review. The thinking activities associated with the five cognitive processes are discussed at length in Section 2.9 Decision-Making Capabilities. SWEG implements mental processing as the servicing and queuing of information, and the pending queue is SWEG's mechanism for servicing a simulation's logic processes. Entries are placed on the queue by the player's initial orders and perceptions, a physical event, or a previous mental event. An individual thought process in SWEG consists of a thinker system, a pending queue entry, and a start and a stop event.

2.11.1 Functional Element Design Requirements

a. SWEG will provide the user with the capability to represent five different aspects of logic processes which are defined as notice, digest, react, ponder, and review.

   1. SWEG will provide the user with the capability to represent physical stimuli to the logic processes. This will be implemented as notice or perceptual recognition.

   2. SWEG will provide the user with the capability to represent the assimilation of stimuli into the memory of the player. This will be implemented as digest or data fusion.

   3. SWEG will provide the user with the capability to represent learned behaviors not requiring careful thinking. This will be implemented as react or resource allocation.

   4. SWEG will provide the user with the capability to introspectively assess the overall perceived situation and change its outlook or attitude accordingly. This will be implemented as ponder or contingency planning.

   5. SWEG will provide the user with the capability to remove outdated perceived information from its memory. This will be implemented as review or information review.

b. If a player type has a subordinate thinker system, SWEG will implement the player's functions of maintaining its pending queue and choosing the next entry for processing. These algorithms are embedded within the source code.

c. SWEG will provide the user with the capability to link thinker systems to sensor receivers so that the logical connection between sensor stimuli and perceptual recognition is maintained.

d. SWEG will maintain the causal relationship and the decoupling between decisions and physical phenomena. Specifically this means that physical events will not be scheduled or caused simply because a decision was made; physical constraints and capabilities will either prevent, allow, or moderate any physical action implied by a decision. The algorithms for this causal relationship and decoupling will primarily be embedded within the source code; however, the results of executing the algorithms can be directly affected by user-defined data representing the physical phenomena.

2.11.2 Functional Element Design Approach

Each player which has at least one subordinate thinker system has a pending queue. Conversely, a player which does not have any thinker systems does not have a pending queue. Should, during the course of an exercise, all of a player's thinker systems be destroyed, that player's pending queue will cease to exist. Any thinker system belonging to a player can process an entry on the pending queue regardless of the platform to which the thinker belongs, but three rules must be followed. First, a thinker can only process the pending queue entry's thinking activity if it has been assigned that thinking activity and a corresponding servicing time in the TIME-TO-THINK table. Second, if the pending queue entry requires the player to notice or digest a sensor chance event, the thinker must be linked to the sensor receiver that produced the sensor chance results. This linkage is made within the TDB player-structure in the LINKAGE data entry. Third, a thinker is limited in the number of items it can process in parallel. The current implementation of SWEG restricts this number to one.

The pending queue can contain 25 different types of entries, and these entries correspond to the 25 thinking activities listed in Table 2.12.1. There is no artificial limit on the number of pending queue entries. Normally, the number of entries will increase if either the servicing times or the intensity of the exercise increases. An increase in exercise intensity usually results in an increase in both the number of messages and sensor results which indirectly causes an increase in the number of resource allocation entries. The capability of a thinker to process a pending queue entry is determined by the thinking activities specified in the TIME-TO-THINK table.

A thought process includes the thinker system, a pending queue entry, and two events which mark the start and stop of the process. A thought process begins when an entry is taken off the pending queue by a thinker, and the process ends when the thinker becomes available to process another pending queue entry. Should the pending queue contain additional entries at the time that a thinker begins a thought process, an incident will be created to report this status. The incident will also report the size of the backlog for each thinking activity which has a pending queue entry.

In SWEG, it is possible for a thought process not to have a stop event. The first instance is the purposeful termination of a thought. A player can choose to stop a thought process during the first event, and an incident is created that reports the abortion of a thought process. The second instance is a non-purposeful termination. If a thinker starts a thought process and is destroyed before it can end the process, the process will not be completed by the thinker or transferred to another thinker. The player will have no memory of all the information associated with the uncompleted thought process and will be unable to complete the thought. No incident is recorded when a thought process is terminated due to the thinker's destruction.

The pending queue is generally a first-in first-out arrangement. Events are added to the pending queue for five reasons: to continue an existing train of thought, to start a new train of thought, to handle the receipt of a stimulus from the physical world, to schedule a wakeup at the end of thinking or as an alarm clock, and to schedule an interrupt for ponder. Four exceptions to the first-in first-out discipline exists. First, wakeups are added to the queue based on the time of the wakeup rather than the time of the origination. Second, sensor chances which are replacing a previous queue entry assume the position of the replaced entry; they do not go to the end of the queue. Third, the addition of an entry from a decision event is placed onto the queue using a push-down stack discipline. Fourth, if a thinker cannot process the first entry on the pending queue, it will continue to look down the queue until it finds an entry it can process or until there are no more entries. A train of thought is a sequence of thought processes that are related by a common theme. For example, noticing and digesting information on a perceived target is a train of thought. Thinking about adding and dropping a target from the lethal engage queue is a train of thought.

The pending queue is implemented in the source code as a leftist tree. This type of structure is semi-sorted; the root or tree's top node has the earliest time; and after it is removed, the two remaining substrees are merged so that the new root node has the earliest time. The rest of the tree is only sorted enough to guarantee the status of the root node. This is an efficient structure, but it means that canceled pending queue entries cannot be removed immediately. A flag is set inside the pending queue entry when the entry is canceled. When the node containing the canceled entry percolates to the top of the tree and is removed, the flag tells the pending queue handler that the entry was canceled and to remove and examine the next root node.

2.11.3 Functional Element Software Design

See Section 2.9.3.

2.11.4 Assumptions and Limitations

- Track correlation is perfect and data fusion is accurate, but may be incomplete.
- Track discrimination is perfect.

2.11.5 Known Problems or Anomalies

None.